Encyclopedia of Database Systems
Springer Science+Business Media, LLC 2009
10.1007/978-0-387-39940-9_848
LING LIU and M. TAMER ÖZSU

Side-Effect-Free View Updates

Yannis Velegrakis[1]

(1) University of Trento, Trento, Italy

Without Abstract

Definition

A view is an un-instantiated relation. The contents of its instance depend on the view query and the instances of the base tables. For that reason, an update issued on the view cannot be directly applied on the view instance. Instead, it has to be translated into a series of updates on the base tables so that when the view query is applied again on the modified base table instances, the result of the view update command will be observed on the view instance. Unfortunately, it is not always possible to find an update translation such that the change observed on the view instance is the one and only the one specified by the view update command. When this happens for a view update translation, the translation is said to have no *side-effects*. To fully exploit the updateability power of views, it is desired to be able to find update translations that have no side-effects.

Historical Background

Updates on the views were introduced almost simultaneously with views. Their importance has been recognized by Codd himself. In fact, one of the twelve rules that Ed Codd [2] introduced to define what a real relational database is, was referring to the ability of the views to be updateable. In particular, the sixth rule was:

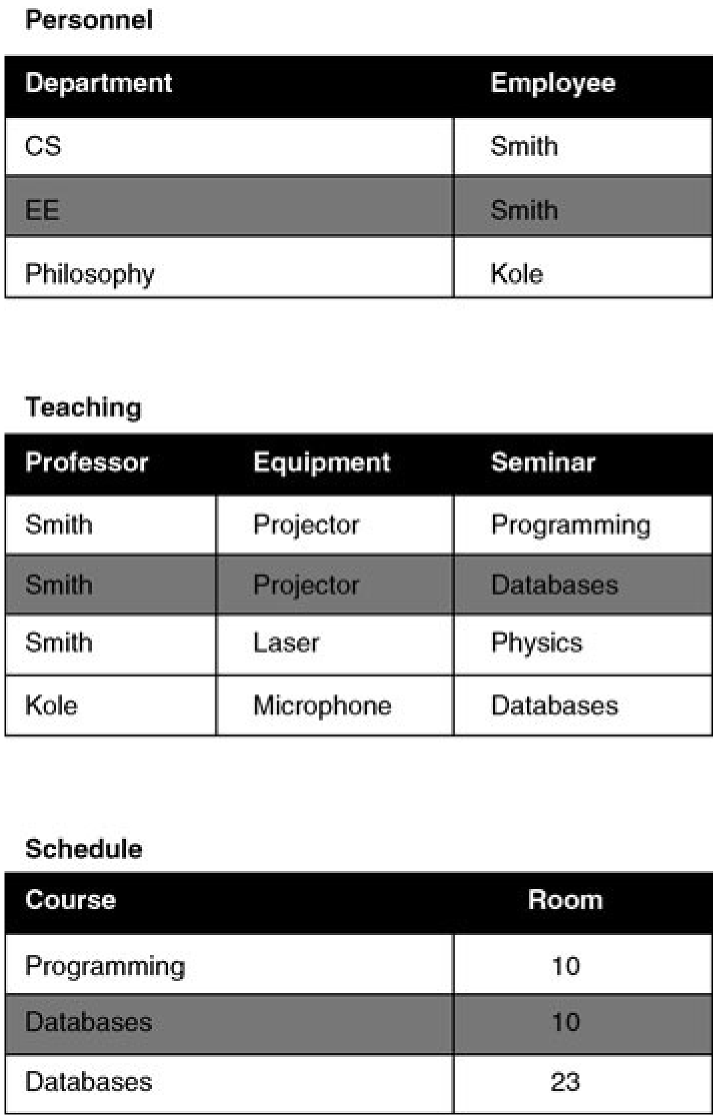
“All views that are theoretically updatable must be updatable by the system”.

The term *theoretically updateable* is referring to the ability of finding side-effect-free translations of the view updates.

Foundations

The problem of side-effect-free updates is based on the problem of *Updates through views*. Referring to Figure 1 of that entry, the update translation W of a view update U on a view V is said to have no side-effects if $U(Q_V(I)) = Q_V(W(I))$, where I is the database instance and Q_V is the view query of the view V.

To better realize the problem of side-effect-free view updates, consider a database instance that consists of the 3 tables of Figure 1.



Personnel

Department	Employee
CS	Smith
EE	Smith
Philosophy	Kole

Teaching

Professor	Equipment	Seminar
Smith	Projector	Programming
Smith	Projector	Databases
Smith	Laser	Physics
Kole	Microphone	Databases

Schedule

Course	Room
Programming	10
Databases	10
Databases	23

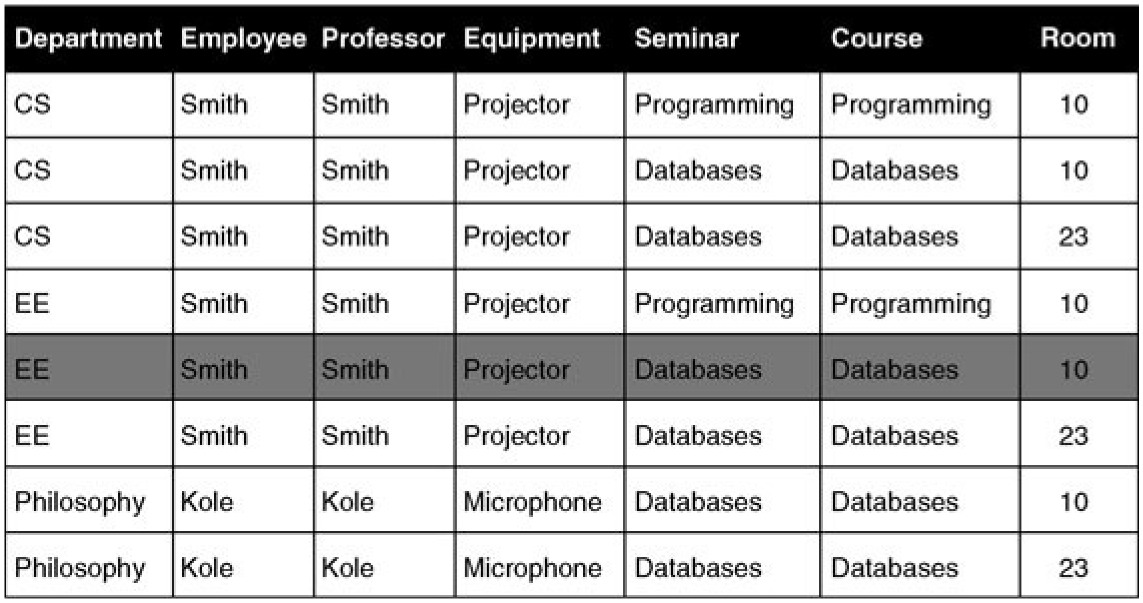
Side-Effect-Free View Updates. Figure 1 Three base Tables.

Suppose that a view V_1 is defined on top of these three tables through the following view query:

```
    select *
from Personnel P, Teaching T, Schedule S
where P.Employee = T.Professor and
T.Seminar = S.Course
```

The instance of the view will be the relation illustrated in Figure 2.

V_1



Department	Employee	Professor	Equipment	Seminar	Course	Room
CS	Smith	Smith	Projector	Programming	Programming	10
CS	Smith	Smith	Projector	Databases	Databases	10
CS	Smith	Smith	Projector	Databases	Databases	23
EE	Smith	Smith	Projector	Programming	Programming	10
EE	Smith	Smith	Projector	Databases	Databases	10
EE	Smith	Smith	Projector	Databases	Databases	23
Philosophy	Kole	Kole	Microphone	Databases	Databases	10
Philosophy	Kole	Kole	Microphone	Databases	Databases	23

Side-Effect-Free View Updates. Figure 2 The view V_1 instance.

Consider now an update command on this view that requests the deletion of the tuple t_d:[EE, Smith, Smith, Projector, Databases, Databases, 10]. Tuple t_d appears in the view instance due to the join of the 3 tuples [EE, Smith], [Smith, Projector, Databases] and [Databases, 10] of the base tables Personnel, Teaching and Schedule. Deletion of any (or all) of these tuples will achieve the desired result of deleting tuple t_d from the view. However, any such deletion will have additional effects in the view instance. For instance, the removal of tuple [EE, Smith] from Personnel will also eliminate the view tuples that are immediate before and after t_d. Similar observations can be made for the tuples in the other two base relations. In fact, for the particular update, it can be shown that there is no change that can be made on the base tables to achieve the desired tuple deletion without any additional changes, i.e., side-effects, in the view instance.

Side-effects are not observed only on deletions but also on insertions. For instance, consider the update command that requests the insertion of tuple t_i:[Economy, Smith, Smith, Projector, Databases, Databases, 10] in V_1. For the appearance of t_i in the view V_1 to be justified, tuples [Economy, Smith], [Smith, Projector, Databases] and [Databases, 10] need to exist in the instances of Personnel, Teaching, and Schedule, respectively. The last two are already there, but not the first. The translation of the insert command on the base tables will insert [Economy, Smith] in Personnel. Unfortunately, due to the value "Smith" in its attribute *Employee*, tuple [Economy, Smith] will be able to join with every other tuple of table Teaching that has value "Smith" in the attribute *Professor*. This will introduce additional tuples in the V_1 instance that the insert command did not request.

Base tables, i.e., tables that have been defined through the "create table" command, have standalone instances, thus, update commands on them can be implemented without side-effects by simply modifying their materialized instance accordingly. If views are to be used as any other table, a view needs to show the same behavior as base tables. This means that update commands need to have translations that generate no side-effects in the view instance. It would have been really surprising for an application or a user that is not aware that a relation she is interacting is actually a view, to request the deletion (or insertion) of a tuple and then see additional tuples disappearing from the view (or appearing in it).

To cope with the view update translation side-effects one option is to leave the burden to the database administrator who defines the view. During the view definition, the administrator is responsible to specify not only the view query but also how exactly each update is translated to updates on the base tables [7] and make sure that side-effects will not occur. The drawback of this option is that it requires a lot of knowledge and experience from the administrators. If the administrator determines that for a given view update there is no translation that has no side-effects, she can make the view not to accept this kind of updates, or allow the side-effects to happen if she believed that this is the semantically correct behavior.

Instead of letting the administrator deciding whether an update should be allowed or not, an alternative solution is to develop methods to perform this test automatically. Based on this idea, Keller [4] developed five criteria to characterize the correctness of a view update translation. The first of these criteria requires the translation to have no side-effects. A consequence of this is that keys of the base relations have to appear in the views, i.e., cannot be projected out. This reduces the cases in which side-effects may appear in the views, but does not completely eliminate them. Keller studied the different choices that exist when translating updates on select, project, select-project and select-project-join views, and provided algorithms for update translation for each case. These algorithms are guaranteed to respect his 5 criteria. For a given update on the view, there may be more than one algorithms that can be used, i.e., more than one possible translations. Which one to be used is a decision that is provided by the database administrator at the moment of view definition [5].

Dayal and Bernstein [3] introduced the notion of the *view-trace* and the *view-dependency* graph. They are graphs that model the dependencies between the attributes of the base tables as determined by the schema and the view definitions. Through them one can determine whether there is an update translation that has no side-effects. For each update on the view, either a unique side-effect-free translation is found and is applied, or the update is not allowed to occur. This approach eliminates the need of an administrator involvement, but cannot be applied in cases in which side-effect generated translations are allowed to occur if they are semantically meaningful.

A different method to determine the kind of updates a view can accept without generating side-effects is through the *constant complement.* Two views are considered complementary if given the state of each view there is a unique corresponding database state. This means that when the instance of one of these views changes (due to an update) while the instance of the other is kept constant, then there is a unique database instance from which the instances of the two views are generated. In other words, the correct translation of the view update is unique [1]. Unfortunately, given a view V, finding its view complement has been shown to be NP-complete even for views with very simple view definition queries.

In summary, it is not always guaranteed that there is a unique side-effect-free translation of a view update. In practical situations, updates are typically allowed on the views. If there are more than one translations of a view update, some criterion may be used to select one of these translations, e.g., the minimality of the changes in the database instance, or some specific parameters that the data administrator specified at the time of view definition. Another approach is to disallow updates on the view that have more than one translation. In the presence of side-effect generating translations, one approach is to allow the translation to take place, allowing that way the side-effects to appear on the view, or to disallow the update completely.

But what would happen in cases in which an update on the view is absolutely necessary, as is the case of an application that can access the database only through a view interface without being aware of the fact that the relation it is accessing is actually a view and with the need to perform updates on it as it would have done if it was a base table? An idea proposed by Kotidis et al. [6] is the following. When an update command is issued on the view, the change in the view instance must be exactly the one described by the update. However, any change on the base table should not take place unless it is implied by the semantics of the view query and the update command. For instance, the deletion of the tuple [EE, Smith] from Personnel as a translation of the delete command for the view tuple [EE, Smith, Smith, Projector, Databases, Databases, 10] mentioned above, would have implied that the reason that the view tuple is deleted is that Smith stopped being affiliated with the EE department. However, neither the semantics of the update command, nor the semantics of the view query imply something like that. Similar claims can be done for tuples [Smith, Projector, Databases] and [Databases, 10] of tables Teaching and Schedule, respectively. For the specific view tuple deletion, the claim is that no change should be observed in the instances of the three base tables, but tuple [EE, Smith, Smith, Projector, Databases, Databases, 10] will be removed from the view instance. This behavior will only be possible if one can accept views whose instances are not exclusively determined by the results of their view queries on the base tables, but also from the update commands that have been issued on them.

Key Applications

Achieving side-effect-free updates on the views is of great importance for systems that provide access to their data through views, but at the same time need to hide from their users or the applications that use the system the fact that they are dealing with views and not actual relations.

Cross-references

Provenance

Updates through Views

Recommended Reading

1. Bancilhon F.B. and Spyratos N.1981. Update Semantics of Relational Views. ACM Trans. Database Syst., 6(4):557–575,
MATH

2. Codd E.F. Is Your DBMS Really Relational? Computer-World, 1985.

3. Dayal U. and Bernstein P.1982. On the correct translation of update operations on relational views. ACM Trans. Database Syst., 8(3):381–416,
MathSciNet

4. Keller A.M. Algorithms for translating view updates to database updates for views involving selections, projections, and joins. In Proc. 4th ACM SIGACT-SIGMOD Symp. on Principles of Database Systems, 1985, pp. 154–163

5. Keller A.M. Choosing a view update translator by dialog at view definition time. In Proc. 12th Int. Conf. on Very Large Data Bases, 1986, pp. 467–474.

6. Kotidis Y., Srivastava D., and Velegrakis Y.2006. Updates through views: a new hope. In Proc. 22nd Int. Conf. on Data Engineering, 2006.

7. Rowe L.A. and Shoens K.A. Data abstractions, views and updates in Rigel. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1979, pp. 71–81.